НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)
"___"_____2020 р.

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «*Програмний комплекс технології автоматизованого виявлення негативних наслідків взаємодії паралельних процесів*»

Виконав : студент *IV* курсу, групи *ІС-61*
Охрименко Дмитро Юрійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *проф., д.т.н., проф. Стеценко Інна Вячеславівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *доц., к.т.н., доц. Тєлишева Тамара Олексіївна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент *професор кафедри ОТ, д.т.н., доц. Клименко Ірина Анатоліївна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)
“___”_____2020 р.

З А В Д А Н Н Я
на дипломний проєкт студенту

Охрименку Дмитру Юрійовичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «Програмний комплекс технології автоматизованого виявлення негативних наслідків взаємодії паралельних процесів»
керівник проєкту *Стеценко Інна Вячеславівна, д.т.н., професор*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “*7*”*травня* 2020 р. №1081-с
2. Термін подання студентом проєкту “*01*”*червня* 2020 року
3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі
2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних
3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв’язання
4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів

5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема структурна варіантів використань

2. Схема структурна діяльності

4. Схема структурна класів програмного забезпечення

5. Схема структурна компонентів програмного забезпечення

6. Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *«13» квітня 2020 року*

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*13.04.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*17.04.2020*	
3.	*Постановка та формалізація задачі*	*19.04.2020*	
4.	*Розробка інформаційного забезпечення*	*21.04.2020*	
5.	*Алгоритмізація задачі*	*25.04.2020*	
6.	*Обґрунтування використовуваних технічних засобів*	*30.04.2020*	
7.	*Розробка програмного забезпечення*	*01.05.2020*	
8.	*Налагодження програми*	*10.05.2020*	
9.	*Виконання графічних документів*	*13.05.2020*	
10.	*Оформлення пояснювальної записки*	*14.05.2020*	
11.	*Подання ДП на попередній захист*	*15.05.2020*	
12.	*Подання ДП на основний захист*	*01.06.2020*	
13.	*Подання ДП рецензенту*	*02.06.2020*	

Студент Дмитро ОХРИМЕНКО

Керівник Інна СТЕЦЕНКО

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 6120.00.000 ПЗ	Пояснювальна записка	50	
3	А4	ДП 6120.01.000 ТЗ	Технічне завдання	7	
4	А4	ДП 6120.02.000 ССВ	Схема структурна варіантів використань	1	
5	А4	ДП 6120.02.000 ССД	Схема структурна ддіяльності	1	
6	А4	ДП 6120.05.000 ССК	Схема структурна компонентів програмного забезпечення	1	
7	А4	ДП 6120.05.000 ССК	Схема структурна класів програмного	1	
			забезпечення		
8	А4	ДП 6120.06.000 КЕ	Креслення вигляду екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	ДП 6120.00.000.00		
Розробн.		Охрименко Д.Ю.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Стеценко І.В					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-61	
Н/контр.		Тєлишева Т.О.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: Програмний комплекс технології автоматизованого виявлення негативних наслідків взаємодії паралельних процесів

Київ – 2020 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з шести розділів, містить 28 рисунків, 4 таблиці, 1 додаток, 7 джерел.

Дипломний проєкт присвячений розробці програмного комплексу технології автоматичного виявлення негативних наслідків взаємодії паралельних процесів.

Ціль розробки: підвищити ефективність тестування паралельних програм за рахунок автоматизації виявлення негативних наслідків взаємодії паралельних процесів.

Задачі розробки:

- проаналізувати існуючі додатки та визначити їх недоліки;
- розробити програмний комплекс на мові програмування Java;
- провести функціональне тестування системи;

У розділі інформаційного забезпечення визначаються вхідні, вихідні дані та структуру масивів даних.

Розділ математичного забезпечення присвячений аналізу мереж Петрі та алгебри процесів для подальшого коретного створення програмного комплексу.

Програмне забезпечення вміщає в собі вибір засобу розробки, вимоги до технічного забезпечення та архітектуру програмного забезпечення.

У технологічному розділі описане випробування програмного продукту та наведене керівництво користувача.

Ключові слова: ПРОЦЕС, ПАРАЛЕЛЬНИЙ ПРОЦЕС, ПРОГРАМНИЙ КОМПЛЕКС, НЕГАТИВНІ НАСЛІДКИ, ПРОГРАМА ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ.

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of six sections, contains 28 figures, 4 tables, 1 appendix, 7 sources.

The diploma project is devoted to the development of a software package of technology for automatic detection of negative consequences of the interaction of parallel processes.

The purpose of development: to increase the efficiency of testing parallel programs due to automation of detection of negative consequences of interaction of parallel processes.

Development tasks:

- Analyze existing applications and identify their shortcomings.
- Develop a software package in the Java programming language.
- Conduct functional testing of the system.

The section of information support defines input, output data, and structure of data arrays.

The section of mathematical software is devoted to the analysis of Petri nets and process algebra for further correct creation of software.

The software includes the choice of the development tool, hardware requirements, and software architecture.

The technology section describes the software product test and provides a user manual.

Keywords: PROCESS, PARALLEL PROCESS, SOFTWARE COMPLEX, NEGATIVE CONSEQUENCES, PARALLEL CALCULATION PROGRAM.

ЗМІСТ

ВСТУП

Дипломний проєкт присвячений створенню програмного комплексу технології автоматизованого виявлення негативних наслідків взаємодії паралельних процесів. Такий програмний комплекс покликаний допомагати розробникам програм паралельних обчислень, адже відловити негативні наслідки дуже складно, бо вони не екрануються.

Наразі у кожній програмі, яка використовується у технологічних фірмах і не тільки, використовують можливість багатопоточності, адже найголовніше, що дає нам автоматизація процесів у вигляді програмного забезпечення – це час, який ми можемо використовувати максимально продуктивно.

При розробці програм, у більше, ніж 50% випадків, трапляються проблеми з коректним виконанням та завершенням програм через недостатній аналіз й відслідковування deadlock, livelock, starvation. Використовуючи запропонований програмний комплекс програмісти зможуть відловити некоректні потоки, через які може не працювати цілісна програма.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Паралельні обчислення – це одночасне використання ресурсів ЕОМ для розв’язування обчислювальних задач. Для цього кожна задача розбивається на підзадачі, які можуть виконуватися у один і той самий момент часу, а інструкції для кожної підзадачі виконуються одночасно на різних процесорах.

Важливість дослідження та використання паралельних обчислень зумовлена наступними причинами:

- явища у реальному світі відбуваються паралельно, тому паралельні обчислення є значно більш придатними для моделювання складних взаємопов’язаних процесів порівняно із послідовними обчисленнями;
- економія часу;
- можливість знаходження розв’язку складних практичних та теоретичних задач.

Процес - це набір коду та даних, які мають спільний VAP. Одна програма зазвичай виконується послідовно в одному потоці, але є винятки (наприклад, сучасні ігри створює окремий потік для графіки, та для функціоналу гри, який приносить певні плюси, наприклад, швидкодію гри та її оптимізацію). Процеси не мають впливу один на одного, тому доступитися до пам'яті іншого процесу ми не можемо (за допомогою допоміжних інструментів виконується взаємодія між потоками).

Для кожного процесу операційної системи створюється ВАП (віртуальний адресний простір), до якого він має доступ. Процесу належить цей ВАП, що вміщає в собі його дані і повністю підвладний процесу. ОС контролює те, як ВАП процесу впливає на фізичну пам'ять.

ОС контролює сторінки пам'яті, які є областями з заданим фіксованим розміром. Якщо пам’яті мало, система надає нові додаткові сторінки пам'яті.

Сторінки віртуальної пам'яті можуть впливати на пам'ять у будь-якому порядку (рисунок 1.1).

Рисунок 1.1 – Схема взаємодії операційної системи з пам’яттю

Під час запуску програми ОС створює процес, завантажуючи програмні дані і код у свій ВАП, а потім запускає основний потік програми.

Одна одиниця виконання коду - це один потік. Кожен потік послідовно виконує вказівки свого процесу, до якого він відноситься, паралельно чи псевдопаралельно з іншими потоками.

Відомо, що одне ядро процесора має один блок виконання в даний момент часу. Це означає, що одноядерний процесор може виконувати команди тільки послідовно, одна за одною. Але також є можливим запуск декількох паралельних потоків у системах, де використовується одноядерними процесори. В такому випадку програма з деякою періодичністю буде перемикатися між потоками, що по черзі дозволить запускати тей чи інший потік. Ця схема називається псевдопаралелізмом. Система запам'ятовує стан кожного потоку перед переходом на інший потік і відновлює його, коли він повертається до виконання потоку. Контекст потоку включає такі параметри, як стек, набір значень реєстру процесора, адреса команди, що виконується, та багато іншого ...

Простими словами, при псевдопараллельном виконанні потоків процесор то виконує один потік, то інший, виконуючи їх почастинно(рисунок 1.2).

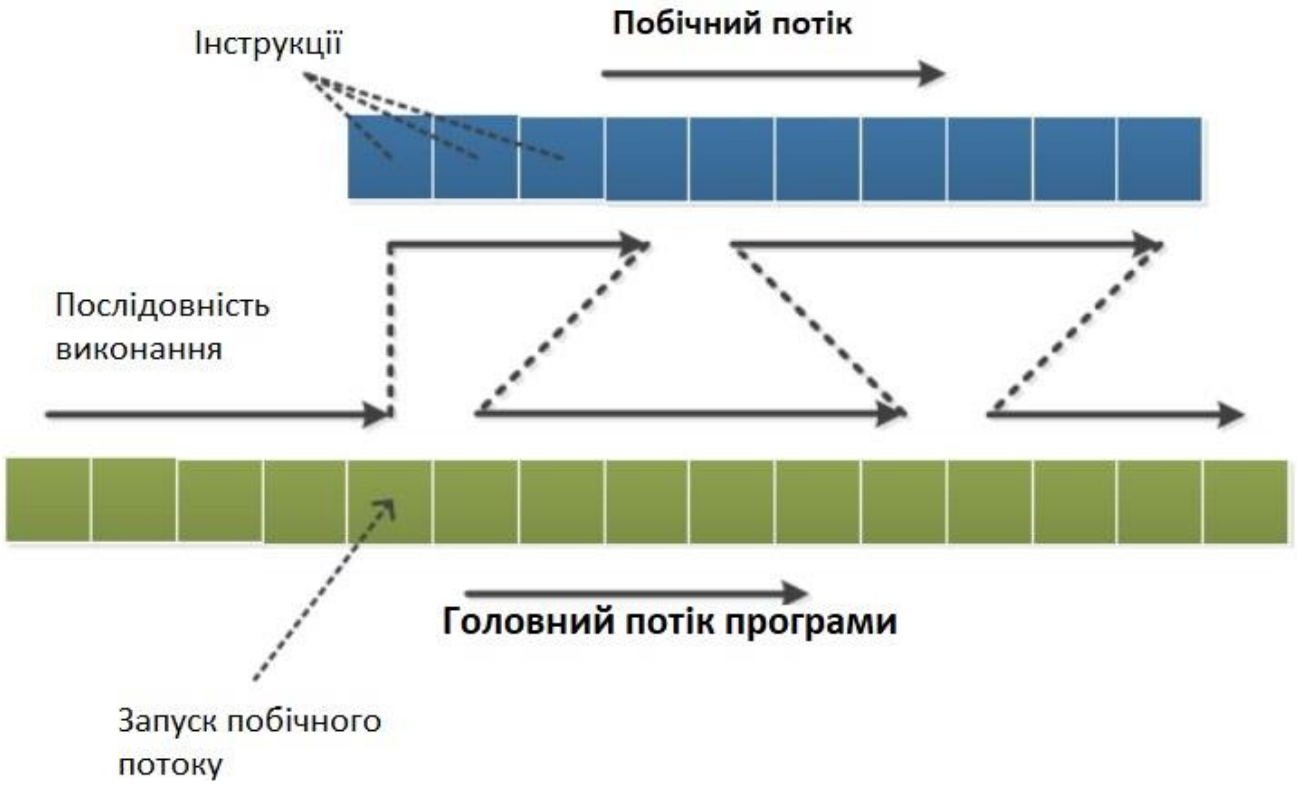

Рисунок 1.2 – Псевдопаралельне виконання потоків

Кольорові квадрати на рисунку - це вказівки процесора (зелені - вказівки основного потоку, сині - побічного). Виконання йде зліва направо. Коли запущений побічний потік, його вказівки починають змішуватися з основними вказівками . Кількість інструкцій, які слід виконати для кожного підходу, не визначається.

Той факт, що інструкції з паралельного потоку виконуються з перервами та упереміш, інколи може призвести до конфліктів потоків[8].

1.1.1 Опис процесу діяльності

Процес виявлення негативних наслідків взаємодії паралельних обчислень ще не є автоматизованим, тому програмісти це роблять мануально. Враховуючи автоматизацію цього процесу, діаграма активності взаємодії користувача із системою виглядатиме наступним чином (рисунок 1.3):

Рисунок 1.3 – Діаграма активності

1.1.2 Опис функціональної моделі

Основним актором у системі є користувач. Він має інтегрувати програмний комплекс в свій код, щоб отримати результат.

Рисунок 1.4 – Діаграма варіантів використання

Відповідно до визначених варіантів використання виявлено функціональні вимоги та встановлена їх пріоритетність, результат наведено у таблиці 1.1.

Таблиця 1.1 – Функціональні вимоги

Актор	Варіант використання	Функціональна вимога	Пріоритет
Користувач	Інтеграція програмного комплексу в власний код	1. Програмний комплекс надає можливість запуску програми	Важливий
		1.1 Програмний комплекс надає можливість перегляду лог файлу	Важливий
		1.2 Програмний комплекс надає можливість редагування уразливих частин коду.	Важливий

1.2 Огляд наявних аналогів

Розглянемо аналоги додатків:

- ***Додаток YourKit java profiler*** – YourKit розробив революційний підхід до профілізації додатків Java як на стадії розробки, так і на стадії виробництва, надаючи незрівнянні переваги професійним розробникам Java[2];
- ***Додаток JProfiler*** – це професійний інструмент для аналізу того, що відбувається всередині працюючої JVM. Ви можете використовувати його в розробці, для забезпечення якості і для завдань пожежогасіння, коли ваша виробнича система має проблеми[3];

– ***Додаток Java VisualVM*** – це спрощений, але надійний інструмент профілювання для додатків Java. За замовчуванням цей інструмент входить в комплект Java Development Kit (JDK). Його робота залежить від інших автономних інструментів, що надаються в JDK, таких як JConsole, jstat, jstack, jinfo і jmap[4];

Розроблюваний додаток матиме назву Threads Analyzer. Наведемо результати порівняння програмного комплексу із уже існуючими аналогами у вигляді таблиці 1.2.

Таблиця 1.2 – Порівняльна таблиця аналогів

	YOURKIT	**JPROFILER**	**JAVA VISUALVM**	**THREADS ANALYZER**
Підтримка всіх програм	+	+	+	+
Підтримка всіх ОС	+	-	-	+
Виведення інформації про потоки	+	+	+	+
Виведення інформації про уразливі частини коду	-	-	-	+
Виведення графіків	+	+	-	-

Як бачимо, усі три додатки допомагають ефективно аналізувати роботу потоків, але мають ряд недоліків, один із яких повністю покриває тема даної дипломної роботи – пошук уразливих частин коду.

1.3 Постановка задачі

1.3.1 Призначення розробки

Призначенням розробки даного програмного комплексу є зменшення часу тестування програм паралельного обчислення та уникненню проблем з коректним виконанням та завершення програм через недостатній аналіз й відслідковування негативних наслідків в програмах паралельного обчислення.

1.3.2 Цілі та задачі розробки

Ціль розробки: підвищити ефективність тестування паралельних програм за рахунок автоматизації виявлення негативних наслідків взаємодії паралельних процесів.

Задачі розробки:

- проаналізувати існуючі додатки та визначити їх недоліки;
- розробити програмний комплекс на мові програмування Java;
- провести функціональне тестування системи.

Висновок до розділу

В даному розділі було розглянуто предметне середовище програмного комплексу. Я описав процес діяльності та функціональну модель програмного комплексу, визначив функціональні вимоги. Було порівняно сучасні аналоги даного додатку. Також визначив цілі та задачі розробки програмного комплексу.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Для розробки програмного комплексу потрібно достатньо велика кількість тестових програм паралельних обчислень, аби перевірити програмний комплекс на ефективність та працездатність. Було вирішено створити три програми паралельного обчислення з різною кількістю потоків та імітацією негативних наслідків взаємодії паралельних процесів.

2.2 Вихідні дані

Результатом роботи програмного комплексу є інформація про уразливі частини коду програми паралельних обчислень та інформація про потоки(стан, тощо) для подальшого аналізу.

2.3 Структура масивів інформації

Вихідні дані будуть у вигляді лог файлу. Якщо інформація про знайдені негативні наслідки програми, то вона буде помічатися типом ERROR. Інформація про потоки буде помічатися типом INFO.

Структура масивів інформації відображені у діаграмі (рисунок 2.1).

Рисунок 2.1 – Структура масивів інформації

Висновок до розділу

В даному розділі мною було визначено які будуть вхідні, вихідні дані та в якому вигляді будуть вихідні дані. Навів структуру масивів інформації.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

На сьогоднішній день для моделювання програм паралельних обчислень використовують Мережі Петрі, але також ще існують алгебри процесів, але вони не такі потужні як мережі Петрі.

Алгебра процесів - це інструмент формального опису, який моделює паралельні системи за допомогою їх алгебри і забезпечує апарат аналізу структури і поведінки моделі. Основоположними розробками в даній області прийнято вважати такі алгебри процесів : Calculus of Communicating Systems (CCS), Communicating Sequential Processes (CSP), Algebra of Communicating Processes (ACP), Performance Evaluation Process Algebra[5].

Мережі Петрі (*МП*) являють собою простий і зручний засіб для моделювання різноманітних розподілених систем і процесів. Ця модель була придумана німецьким вченим Карлом Петрі в 1939 році для опису хімічних процесів. Офіційна історія мереж Петрі почалася в 1962 році, коли Карл Петрі захистив дисертацію «Kommunikation mit Automaten », в якій їм і було введено поняття мережі Петрі.

3.2 Математична постановка задачі

Нехай існує деяка предметна область, у конкретному випадку маємо програму паралельних обчислень.

Задача: розробити програмний комплекс на основі аналізу двух способів моделювання паралельних процесів, який буде автоматизовано виявляти негативні наслідки взаємодії паралельних процесів.

3.3 Обґрунтування методу розв’язання

Для розв’язання задачі розробки програмного комплексу, треба проаналізувати два методи моделювання паралельних процесів:

- мережі Петрі;
- алгебра процесів;

3.4 Опис методів розв’язання

Мережі Петрі

Мережі Петрі являють собою двочастковий орієнтований граф, що містить вершини двох типів - місця (позначаються кружками) і переходи (позначаються прямокутниками). Будь-яка дуга веде або від вершини-місця в вершину-перехід, або навпаки. Дуги, що з'єднують два місця або два переходи, заборонені. Місця, у які немає вхідних дуг, називаються вхідними. Місця, у яких немає вихідних дуг, називаються вихідними.

Кожне місце мережі Петрі може містити нуль або більше міток (маркерів, англ. Tokens). Всі мітки вважаються однаковими і не відрізнятись один від одного. Розподіл міток по місцях мережі називається її розміткою. Робота мережі починається з початкової розмітки.

Мітки можуть переноситися з одного місця на інше. Перенесення міток виконується за наступними правилами:

- перехід є активним, якщо кожне його вхідний місце містить принаймні одну мітку (точніше - по одній мітці на кожну що входить в цей перехід дугу);
- активний перехід може спрацювати, при спрацьовуванні перехід поглинає по одній мітці з кожного свого вхідного місця і розміщує по одній мітці на кожне своє вихідне місце (по одній мітці на кожну вихідну дугу);
- у кожен момент часу для спрацьовування з усіх активних переходів недетермінованим чином вибирається один з них. Якщо активних переходів немає, то робота мережі на цьому завершується;

Позначимо кожний перехід мережі Петрі деяким унікальним символом (наприклад пронумеруємо їх). Послідовність символів σ, в якій кожний i-ий символ такий же як символ переходу, спрацьованого на і-ому кроці мережі, називається послідовністю спрацювання мережі (рисунок 3.1). Послідовність спрацювання визначає послідовність розміток μ_i, де μ_0 рочаткова розмітка.

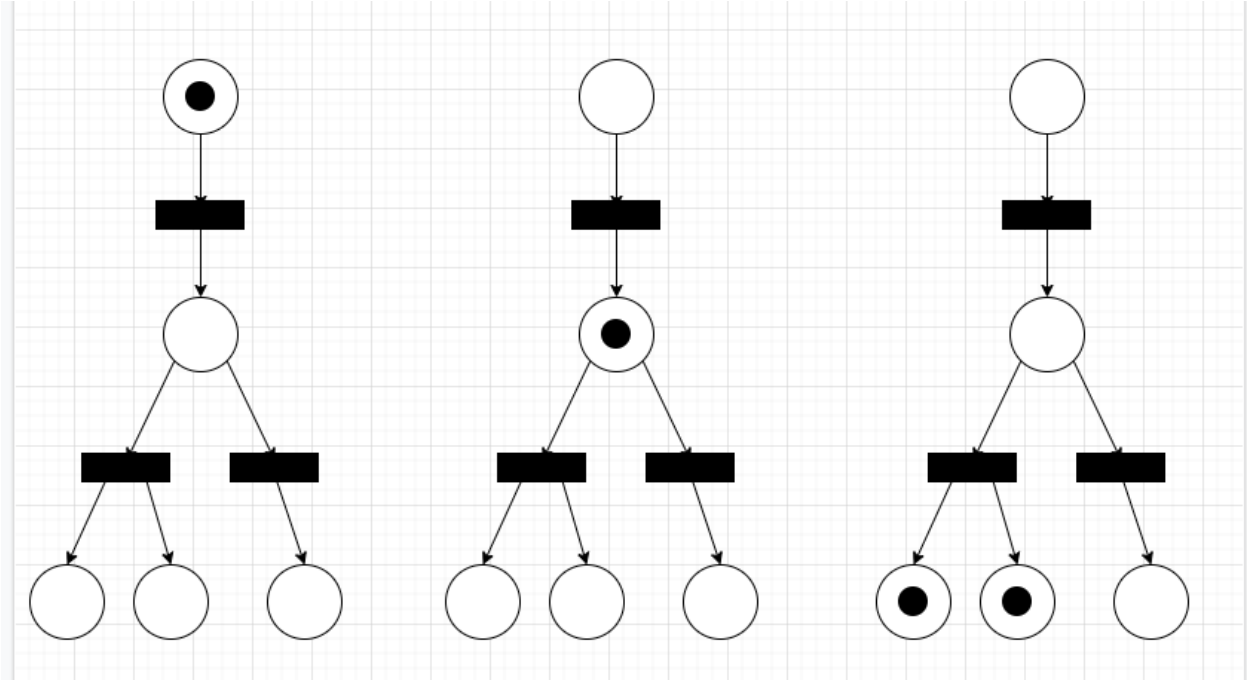

Рисунок 3.1 – Приклад послідовної роботи мережі

Білими колами позначені місця, чорними прямокутниками — переходи, чорними колами — мітки.

Окремі елементи мережі Петрі (місця та переходи) можуть мати різні властивості, основуючись на них спочатку визнаються властивості мережі, а все потім її класифікація.

Найпростішою властивістю місць є кількість міток, які можуть поміститися в це місце. Якщо в любій досяжній розмітці кількість міток в заданому місці буде не більше одної (0 або 1) то таке місце називають безпечним. А мережі Петрі називають безпечними, коли всі місця безпечні. В безпечних мережах стан кожного місця описують лише одним бітом, з цього може зробити висновок, що такі мережі можуть бути реалізовані апаратно,

використовуючи різні види переключателів (тригерів). Доречи, першим варіантом визначення мережі Петрі, яке дав сам Адам Петрі, було те що мережа є безпечною.

Однак, для більшості додатків вимога безпеки мережі дуже строга. Її можливо послабити, якщо дозволити кожному місцю зберігати деяку обмежену кількість міток. Місце буде називатися k-обмеженим, якщо в будь-якій досяжній розмітці в даному місці буде не більше ніж k міток. Очевидно, що 1-досяжна місце являється безпечним. Місце є обмеженим, якщо існує таке k, що це місце являється k-обмеженим. А ось мережа Петрі являється k-обмеженою, якщо будь-яке місце є k-обмеженим, та просто обмеженою, якщо всі її місця обмежені. Обмежені мережі також мають апаратну реалізацію, в якій кожне місце існує як лічильник (наприклад, регістр) з заданим розміром. Необмежені мережі несуть тільки теоретичну зацікавленість.

Ще одною властивістю мереж є властивість консервативності, яка базується на підрахуванні міток. Мережа називається консервативною, якщо кількість міток в будь-якій досяжній розмітці залишається незмінним (еквівалентним початковій розмітці). Така модель використовується наприклад, в тих випадках, коли мітки являють собою деякі ресурси системи, які не знешкоджуються і не створюються. Ці ресурси можуть переходити з однієї частини системи в іншу, але їх загальна кількість в процесі роботи системи не змінюється. Не тяжко показати, що будь-який перехід, який спостерігається хоча б в одній досяжній розмітці, повинен мати однакову кількість вхідних та вихідних дуг, тобто скільки він взяв міток, стільки ж він має їх і поставити.

В стандартній інтерпретації мереж Петрі, кожен перехід трактується як деякий процес, який бере свої вхідні дані із вхідних місць, робить обробку цих даних та кладе результати в свої вихідні дані. Однак, так сталося, що тривалість цих процесів ніяк не конкретизується. Хоча мережі Петрі мають необхідний потенціал для моделювання паралельних процесів, але очевидно

що розмовляти про паралельні процеси маємо сенс тільки тоді, коли вони мають деяку тривалість.

В найпростішому варіанті мереж Петрі може розглядатися послідовний пристрій, робота якого закладається у тому, що він реалізує деяку послідовність спрацювань. Така інтерпретація має сенс в декількох випадках:

- якщо в моделі передбачається неперервний час, а тривалість спрацьовування переходів буде нульовим, то ймовірність одночасного спрацювання двух переходів можемо вважати нульовою. Очевидно, що в даному випадку, важливий якраз і є порядок (послідовність) спрацювання переходів.
- робота мережі Петрі застосовується, коли всі процеси (переходи) обробляються одним і тим же пристроєм. Прикладом такого роду системи може бути одно процесорний комп'ютер, в якому виконується одночасно декілька різних операцій. Ці процеси можуть бути логічно не зв'язані друг з другом (та розглядатися користувачем як паралельними), але реалізуються ці операції строго послідовно один за другим.

Більш ефективними в багатьох випадках є інтерпретації мереж Петрі, в яких процеси можуть оброблятися одночасно. У стандартній моделі такої мережі передбачається, що час спрацювання всіх переходів є однаковим. Для простоти будемо рахувати, що цей час дорівнює одиниці. Також припустимо, що спрацювання будь-якого переходу виконується в цілочисельний момент часу. Таким чином, час в моделі виявляється дискретним, та його відлік буде виконуватися з нульового моменту часу (початкова розмітка).

Для того, щоб описати паралельну поведінку мережі, потрібно визначити поняття конфлікту. Конфліктну в мережі Петрі називають ситуацію, в якій відразу декілька активних переходів хочуть взяти одну й ту саму мітку деякого місця (рисунок 3.2). При послідовному спрацюванні

переходів ніякі конфлікти не виникають, однак при паралельній інтерпретації потребується деякий спосіб їх вирішування.

Рисунок 3.2 – Приклад конфлікту в мережі Петрі

Стандартна схема паралельної обробки переходів в мережах Петрі виглядає наступним чином:

- із всіх активних в даний момент часу переходів, обирається ті, в яких відсутні конфлікти (тобто ніякі два обрані переходи не мають взаємного конфлікту);
- всі переходи спрацьовують одночасно;

– та якщо немає активних переходів, то мережа закінчує свою роботу;

Недетермінізм мережі, закладається у тому, що у виборі переходів для спрацювання, можна суттєво понизити, якщо впровадити правило, яке називається принципом максимального паралелізму: на кожному кроці для спрацювання обирається така підмножина A' множини A активних переходів, що:

– ніякі два переходи з A' не конфліктують;
– для будь-якого не обраного переходи з множини $A \backslash A'$ є хоча б один конфліктний перехід в A'. Іншими словами, обирається такий набір переходів, який не може бути розширений без конфліктів ніякими другими переходами. Чи буде такий набір найбільшим (по кількості переходів) не важливо, головне, щоб він був не розширюваним;

За допомогою мереж Петрі можливо відобразити створення потоків їх запуск та завершення (рисунок 3.3).

```java
public class ApplicationRunner {
    public static void main(String[] args) {
        Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.print("HELLO");
            }
        });
        thread.start();
    }
}
```


Рисунок 3.3 – Створення потоків, їх запуск та завершення

В даний момент часу існує велика кількість варіацій мереж Петрі, які одним чи другим способом розширюють, або звужують функціонал моделі. Приклади таких:

– мережа з пріоритетом – це стандартна мережа, де кожному переходу t, якому поставлено деяке число Pr t, зване пріоритетом переходу (рисунок 3.7). Пріоритети використовуються для більш повного управління вирішення конфліктних ситуацій. Якщо два переходи t_1 та t_2 конфліктують через деякий спільний ресурс, то перевагу отримає той, в якого пріоритет вище. Якщо всі переходи даної мережі будуть мати різні пріоритети, то в такій системі взагалі не буде конфліктів. На практиці достатньо визначити лише декілька пріоритетів для потенційно конфліктних переходів:

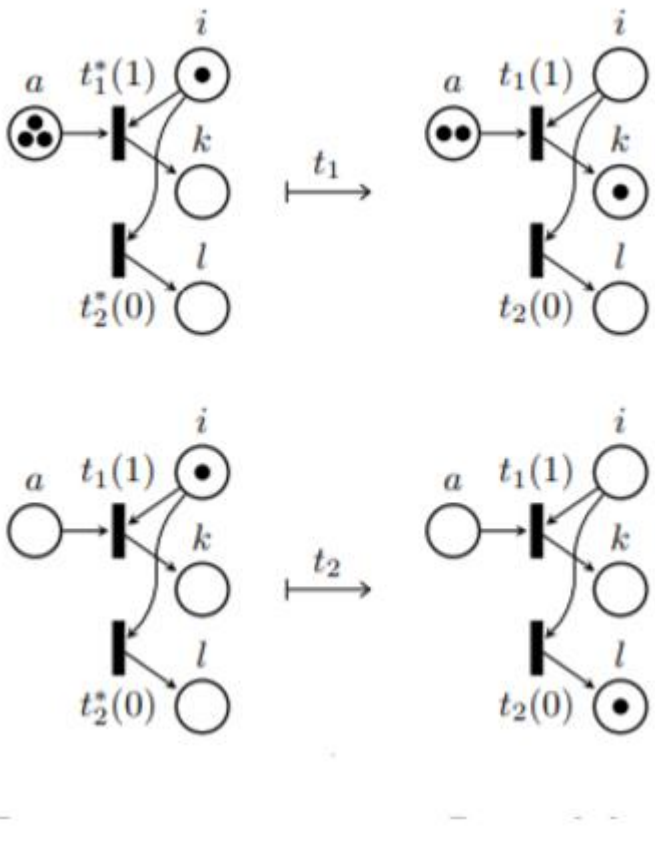

Рисунок 3.7 – Приклад роботи мережі з пріоритетом

Мережа має два переходи t_1 та t_2 з пріоритетами Pr $t_1 = 1$ та Pr $t_2 = 0$. Якщо місце a має хоча б одну мітку, то активними є два

переходи, причому вони конфліктують через одну мітку в місці i. Так як пріоритет $t_1 \geq t_2$, то саме він і спрацює. Якщо ж місце а пусте, то активним буде тільки перехід t_2 і спрацює саме він.

- інгібіторні мережі. В інгібіторних мережах Петрі до стандартних дуг, які ведуть з місць в переходи, додаються інгібіторні дуги (сповільнюючи дуги). Така дуга, при наявності в правильному місці хоча б одної мітки, перешкоджає спрацюванню переходу. В цього випливає, що цей перехід спрацює лише тоді, коли в цьому місці закінчяться всі мітки. Вони створені для того, щоб перевіряти наявність міток в заданому місці;
- кольорові мережі Петрі. В стандартних мережах всі мітки завжди однакові. Однак бувають системи, які моделюють мережами Петрі, мітки часто представляють собою різні об'єкти. В кольорових мережах Петрі кожна мітка має свій колір, що дає можливість відрізняти одну мітку від іншої. Переходи в кольорових мережах визначають відношення між вхідними мітками та вихідними мітками. Для цього вхідним дугам кожного переходу приписуються передумови, які визначають, з якими значеннями мітки поглинаються цим переходом. Вихідні дуги переходу визначають значення міток, які будуть розташовані в відповідні місця;
- часові мережі Петрі. Для моделювання системи, в якій окремі підпроцеси мають різну тривалість, ми маємо змогу використовувати часові мережі. В таких мережах кожна мітка отримує додатковий атрибут – час затримки (timeout). Якщо мітка з'явилася в якомусь місці в момент часу t, то вона буде доступною для всіх переходів (зв'язаних з даним місцем) починаючи з моменту $t + \tau$. Затримки, які призначаються міткам, приписують

дугам, які ведуть від переходів. Отже, всі мітки, які йдуть по цій дузі будуть мати однакову затримку, яка приписана цій дузі;

Алгебри процесів

Все алгебри включають опис активних компонент і механізми взаємодії між ними, в більшості випадків активні компоненти представлені процесами або агентами, що складаються з активностей, кожна з яких задає певні дискретні дії системи, будь-яка активність може бути внутрішньої по відношенню до процесу, або ставити взаємодію між процесами, така взаємодія носить бінарний характер, тому кожна активність $\alpha \in \Delta$ завжди має пов'язану активність $\bar{\alpha} \in \bar{\Delta}$, і це саме та сполучена активність, з якою готова синхронізуватися активність α.

В алгебрі процесів можна приховати поведінку фрагмента процесу, представивши його як чорний ящик. При цьому виникає можливість побудувати складну імітаційну модель, виключивши більш прості частини, поведінка яких не представляє інтересу при дослідженні. Невидиму поведінку моделюють з допомогою спеціальної активності σ, представляє внутрішні дії процесу. Таким чином, повний алфавіт активностей $Act = \wedge \cup \{\sigma\}$ включає активності взаємодії $\wedge = \Delta \cup \bar{\Delta}$ та множина $\{\sigma\}$ внутрішніх активностей.

Найпростіший константний процес не може виконувати жодної активності і призводить до звільнення всіх ресурсів за допомогою виконання активності nil. Якщо процес включає більше однієї активності, послідовність їх звершення регулюється префиксной записом $\alpha.P$, сенс якої полягає в тому, що вона представляє процес, спочатку виконує активність α, а потім функціонує, як P. Прикладом елементарного процесу P_1 може служити вираз:

$$(1)\, P_1 \coloneqq begin.end.nil.$$

Дії процесу згідно з цим висловом складаються з прямої послідовності трьох активностей. Активність $begin$ виконує дії, пов'язані з початком функціонування процесу P_1, end завершує процес P_1, nil звільняє ресурси та видаляє процес P_1 із моделі.

Для моделювання систем з паралельним функціонуванням компонентів синтаксис алгебри доповнюють операціями вибору. Операція $P + Q$ представляє процес, який чекає готовності процесів P або Q взяти участь в деякій активності. Після цього відбувається недетермінований вибір між ними. Застосувавши префіксних запис, наведемо приклад процесу P_4, використовуючого дану операцію:

$$(2) P_4 \coloneqq begin_1.P_1 + begin_2.P_2$$

Процес P_4 передбачає вибір між двома альтернативами. Цей вибір контролюється початковими активностями складових його процесів і не залежить від зовнішнього впливу.

Для того щоб конструювати складні процеси, необхідно також вміти паралельно запускати кілька підпроцесів, забезпечуючи механізми взаємодії і синхронізації. З цією метою використовують паралельну конструкцію $P||Q$, яка представляє два процеси, що розвиваються паралельно. Процесам P і Q дозволено будь індивідуальну поведінку, однак при наявності в них пов'язаних активностей буде синхронізуватися шляхом одночасного звершення активностей.

До загальноприйнятих операціями, що використовуються в різних алгебрах процесів, відносять перейменування $P[K]$ і стиснення $P\backslash A$. Процес $P[K]$ поводиться як P, але у всіх своїх активностях перейменовується функцією K. Процес $P\backslash A$ поводить себе як P, але не дозволяє виконувати ніяких активностей з A.

Розглянуте неформальне описання основних синтаксичних операцій, характерних для різних алгебр процесів, дає загальне уявлення про цей інструмент формалізації. Висока ефективність даного підходу стимулює розробників застосовувати алгебри процесів до самих різних моделей. На сьогоднішній день добре розроблені підходи до використання алгебри процесів для побудови стохастичних моделей паралельних структур і для імітаційних моделей з синтетичними робочими навантаженнями, що

базуються на імовірнісному розподілі параметрів вхідних завдань. Однак такі підходи вимагають попередніх знань про систему і її робочому навантаженні для отримання адекватних результатів моделювання. При використанні моделей на етапі розробки складної системи такі знання., Як правило, відсутні. Тому становить значний інтерес створення алгебри процесів для імітаційного моделювання систем з реальної робочої навантаженням.[5]

Висновок до розділу

Сучасні засоби моделювання широко використовують об'єктно-орієнтований підхід, який в поєднанні з візуальним представленням інформації дозволяє суттєво скоротити час побудови моделі складної системи. Однак надмірна характер візуального опису та неточна семантика часто стають нездоланними перешкодами для побудови таких моделей, еквівалентність яких об'єкту моделювання повинна бути строго обґрунтованою. Рішення проблеми лежить у сфері використання низькорівневих засобів формального опису, що дозволяють застосовувати принципи визначення еквівалентності, які використовуються в загальній теорії систем[5]. Засоби такого типу: мережі Петрі, алгебри процесів. Мережі Петрі більш потужніші ніж алгебри процесів, тому на сьогоднішній день використовують мережі Петрі.

Отже, при розробці програмного комплексу, ми маємо більше приділити уваги аналізу мереж Петрі.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Для реалізації програмного комплексу автоматизованого виявлення негативних наслідків взаємодії паралельних процесів, застосовують мови програмування, які маю підтримку багатопоточності та багатий інструментарій для роботи з потоками. Для написання даної дипломної роботи було прийняте рішення обрати програмну мову Java. Основною перевагою цієї мови є те, що вона розроблена з підтримкою паралельних обчислень, і всі обчислення виконуються у контексті потоку. Декілька потоків можуть спільно використовувати об'єкти і ресурси; кожен потік може виконувати свої інструкції (код), але потенційно може отримати доступ до будь-якого об'єкта в програмі. Відповідно, як наслідок, існує безліч механізмів та бібліотек для організації багатопоточної програми, аби можна було обрати найбільш зручний спосіб для вирішення конкретної задачі паралельних обчислень.

Деякі переваги обраної мови програмування:

- це об’єктно-орієнтована мова, ООП більш ефективно організовує структуру програм (особливо великих) та значно спрощує обслуговування та модернізацію старого коду;
- мова високого рівня з простим синтаксисом і плавною кривою навчання;
- стандарт для корпоративних обчислювальних систем;
- кроссплатформеність: Java програми запускаються на будь-якій платформі та у будь-якому браузері, сумісному з Java. Це дає можливість легко переходити з однієї комп'ютерної системи в іншу;
- автоматичне управління пам’яттю та збирач сміття;
- стабільність і велике співтовариство;
- підтримка старих версій Java;

Основою для написання програмного комплексу став інтерфейс ThreadMXBean, цей інтерфейс є інтерфейсом управління JVM(Java Virtual Machine) [7].

Для створення двух розкладів, один з яких запускає сервіс для виявлення негативних наслідків кожні 5 секунд та один, який кожну секунду запускає сервіс для збору даних про потоки в програмі паралельних обчислень був обраний інтерфейс ScheduledExecutorService, який виконує асинхронний код в одному або декількох потоках за розкладом[8].

Для виведення інформації було вибрано лог файл, який буде знаходитись у директорії програми паралельних обчислень в яку буде інтегровано програмний комплекс, тобто якщо це навіть клієнт-серверний додаток то файл буде розташований на сервері. Для реалізації лог файлу було використано допоміжну бібліотеку Java log4j, яка є надійною, швидкою та гнучкою середовищем ведення журналів[9].

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Мінімальні вимоги до характеристик комп’ютера:

- оперативна пам’ять – 8 Гб;
- процесор – Intel Core i5 2,9 МГц;
- графічна карта – Nvidia GeForce GTX 660;

Програмні засоби, що використовуються для проведення тестування:

- операційна система Windows 10;
- IDE Idea;
- вихідний код програми паралельних обчислень.

Технічні засоби, що використовуються для проведення тестування:

- оперативна пам’ять – 16 Гб;
- жорсткий диск – 500 Гб;

– процесор – Intel Core i5 8th Gen;

– графічна карта – Nvidia GeForce GTX 780;

4.3 Архітектура програмного забезпечення

4.3.1 Діаграма класів. Додайте текстовий опис файлів

Діаграма класів описує архітектуру програмного комплексу (рисунок 4.1).

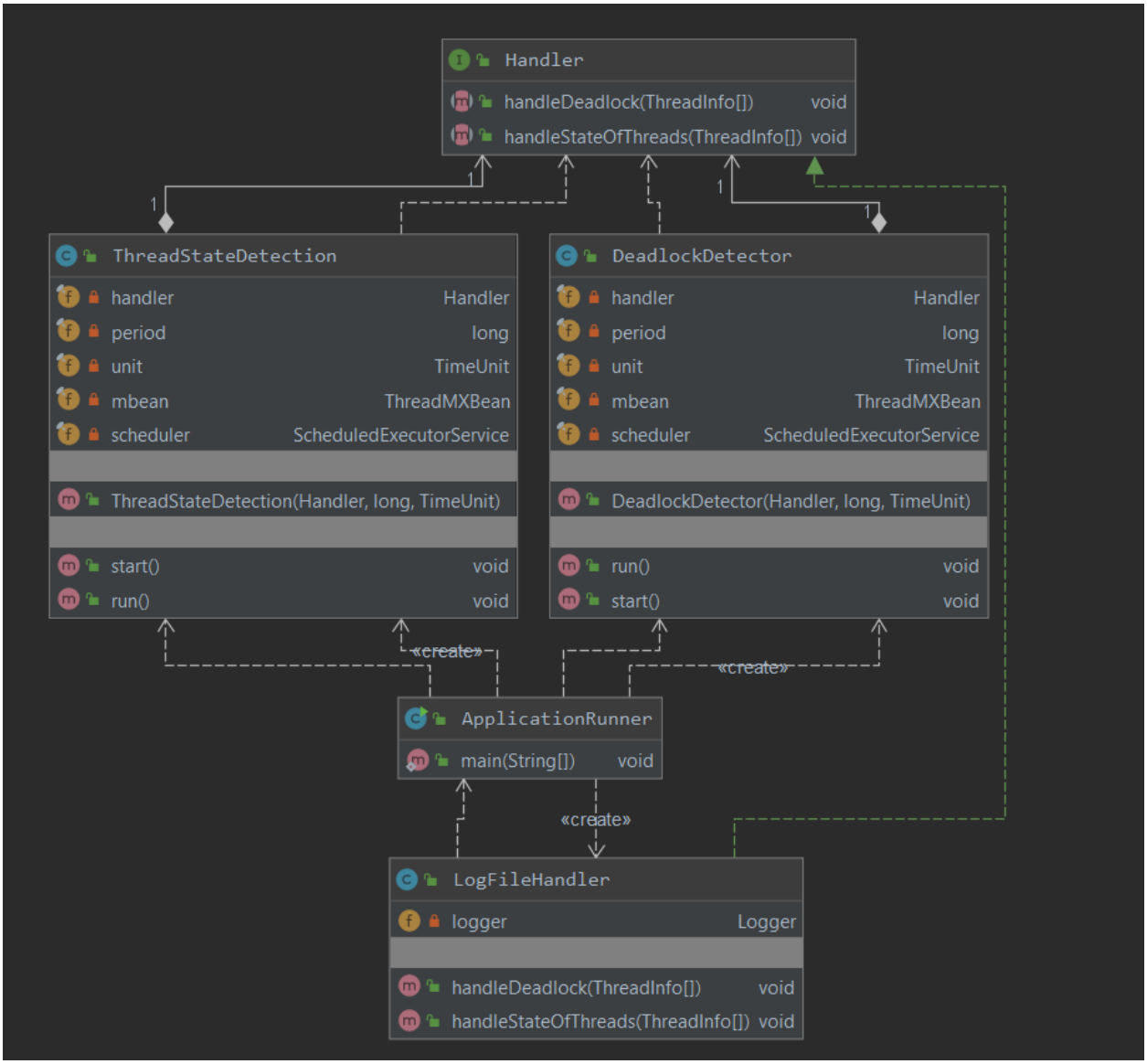

Рисунок 4.1 – Діаграма класів

4.3.2 Діаграма послідовності

Діаграма послідовності описує процес роботи програмного комплексу (рисунок 4.2)

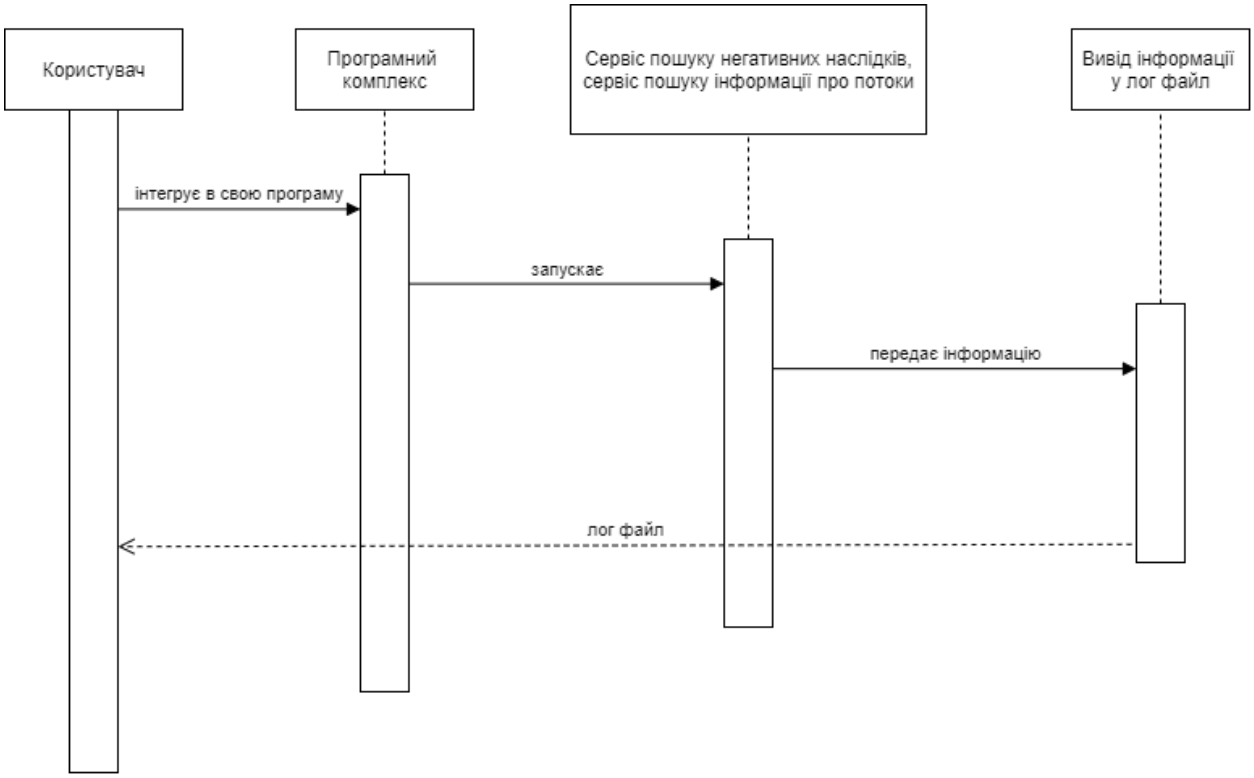

Рисунок 4.2 – Діаграма послідовності

4.3.3 Діаграма компонентів

Діаграма компонентів описує робото програмного комплексу з програмою паралельних обчислень та комп’ютером (рисунок 4.3).

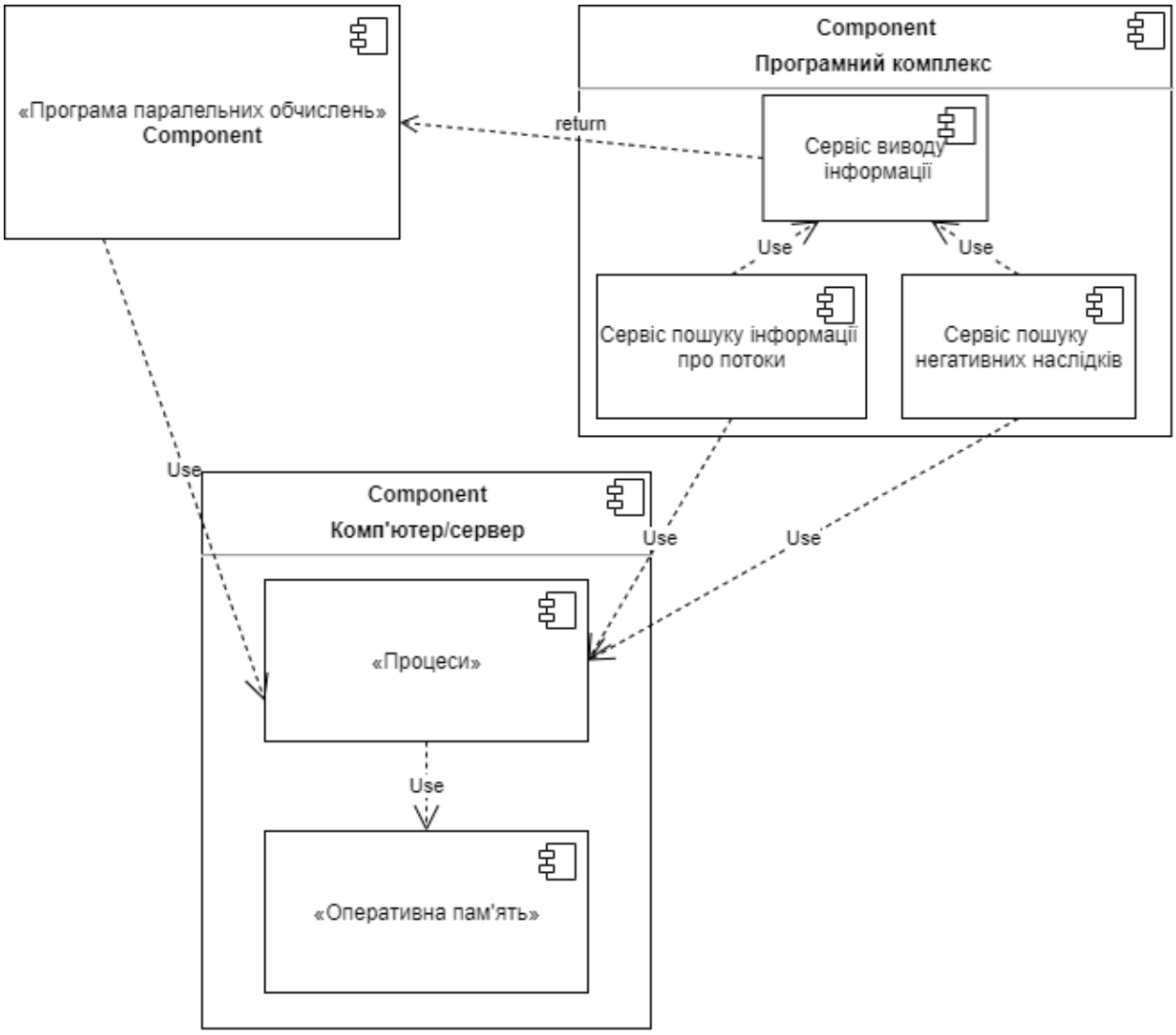

Рисунок 4.3 – Діаграма компонентів

4.3.4 Специфікація функцій

Опис функцій у таблиці 4.1.

Таблиця 4.1 – Специфікація функцій

Назва функції	**Опис функції**	**Вхідні дані**	**Вихідні дані**
DeadlockDetector.run()	Шукає негативні наслідки взаємодії паралельних процесів	Немає	Немає
start()	Запускає розклади	Немає	Немає
scheduleAtFixedRate(this, this.period, this.period, this.unit)	Запускає процеси за розкладом	Об’єкт сервісу, проміжок між запусками, період, одиниці вимірювання часу	Немає
getThreadMXBean()	Вертає екземпляр класу ThreadMXBean	Немає	Екземпляр класу ThreadMXBean

Назва функції	**Опис функції**	**Вхідні дані**	**Вихідні дані**
newScheduledThreadPool(1)	Дозволяє лише одному потоку одночасно за розкладом	Кількість потоків	Немає
findDeadlockedThreads()	Шукає заблоковані потоки	Немає	Ідентифікатори заблокованих потоків
getThreadInfo(deadlockedThreadIds)	Шукає інформацію про потоки	Ідентифікатори заблокованих потоків	Масив інформації про заблоковані потоки
handleDeadlock(threadInfos)	Виводить інформацію про заблоковані потоки	Масив інформації про заблоковані потоки	Немає
ThreadStateDetection.run()	Шукає інформацію про всі потоки, крім заблокованих та системних	Немає	Немає

Назва функції	Опис функції	Вхідні дані	Вихідні дані
handleStateOfThreads(threadInfos)	Виводить інформацію про всі потоки, крім заблокованих та системних.	Масив інформації про всі потоки	Немає

4.4 Опис звітів

Результатом роботи є створений лог файл, в якому присутній аналіз коду, якщо існують негативні наслідки в програмі паралельних обчислень, та профайлінг потоків.

Висновок до розділу

В даному розділі мною було вибрано засіб розробки – програмна мова Java, тому що вона має підтримку багатопоточності та багатий інструментарій для роботи з потоками. Також були опрацьовані вимоги до технічного забезпечення та наведена архітектура програмного забезпечення, та описанні функції програмного забезпечення.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Перш за все користувач має інтегрувати програмний комплекс у свою програму.

1. Створити пустий пакет у своєму проєкті (рисунок 5.1)

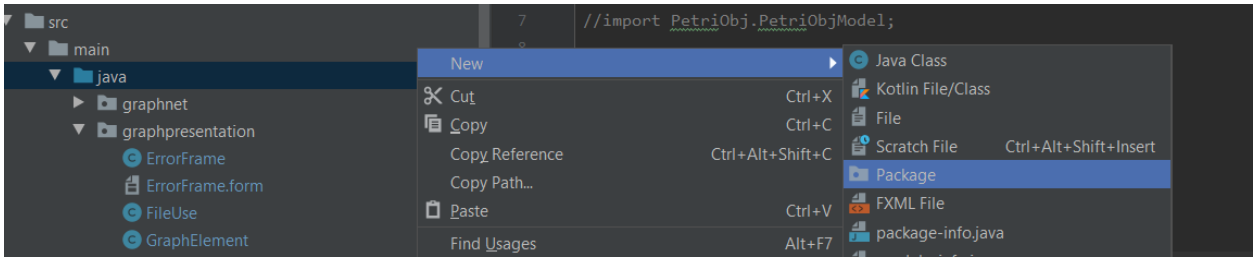

Рисунок 5.1 – Створення пустого пакету

2. Перенести всі файли у цей пакет (рисунок 5.2).

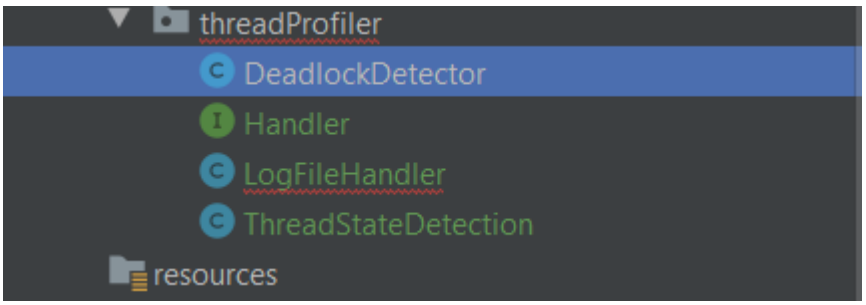

Рисунок 5.2 – Перенесення файлів

3. У папку resources добавити файл log4j.properties (рисунок 5.3)

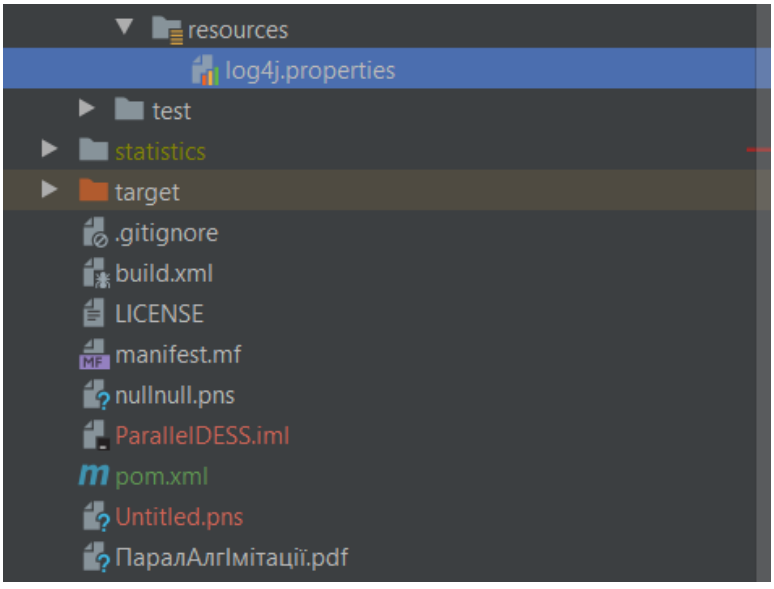

Рисунок 5.3 – Додавання файлу log4j.properties

4. Додати залежність у файл pom.xml (рисунок 5.4).

Рисунок 5.4 – Додавання залежності у файл pom.xml

5. Додати запуск сервісів у метод main() свого проєкту (рисунок 5.5).

Рисунок 5.5 – Додавання запуску сервісів

6. Запустіть свою програму паралельних обчислень та почекайте секунд 30.
7. Потім з’явиться лог файл – logging.log у корневій папці проєкту (рисунок 5.6).

Рисунок 5.6 – Розташування лог файлу

8. Далі зайшовши у цей файл ви побачите всю інформацію (рисунок 5.7)

```
21-05-2020 17:58:28 ERROR : Deadlock detected!
21-05-2020 17:58:28 ERROR : "Thread-0" prio=5 Id=16 BLOCKED on java.lang.String@7c1452d2 owned by "Thread-1" Id=17
21-05-2020 17:58:28 ERROR :     app//ua.kpi.fict.ApplicationRunner$1.run(ApplicationRunner.java:72)
21-05-2020 17:58:28 ERROR : "Thread-1" prio=5 Id=17 BLOCKED on java.lang.String@1a3276f1 owned by "Thread-0" Id=16
21-05-2020 17:58:28 ERROR :     app//ua.kpi.fict.ApplicationRunner$2.run(ApplicationRunner.java:89)
21-05-2020 17:58:34 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:58:35 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:58:36 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:58:37 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:58:38 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:58:38 ERROR : Deadlock detected!
21-05-2020 17:58:38 ERROR : "Thread-0" prio=5 Id=16 BLOCKED on java.lang.String@36f28163 owned by "Thread-1" Id=17
21-05-2020 17:58:38 ERROR :     app//ua.kpi.fict.ApplicationRunner$1.run(ApplicationRunner.java:72)
21-05-2020 17:58:38 ERROR : "Thread-1" prio=5 Id=17 BLOCKED on java.lang.String@65068b5c owned by "Thread-0" Id=16
21-05-2020 17:58:38 ERROR :     app//ua.kpi.fict.ApplicationRunner$2.run(ApplicationRunner.java:89)
21-05-2020 17:59:44 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:59:45 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:59:46 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:59:47 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:59:48 INFO  : "Thread deadlock detector" prio=5 Id=14 RUNNABLE
21-05-2020 17:59:48 ERROR : Deadlock detected!
21-05-2020 17:59:48 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:59:48 ERROR : "Thread-0" prio=5 Id=16 BLOCKED on java.lang.String@6116879a owned by "Thread-1" Id=17
21-05-2020 17:59:48 ERROR :     app//ua.kpi.fict.ApplicationRunner$1.run(ApplicationRunner.java:72)
21-05-2020 17:59:48 ERROR : "Thread-1" prio=5 Id=17 BLOCKED on java.lang.String@79f33297 owned by "Thread-0" Id=16
21-05-2020 17:59:48 ERROR :     app//ua.kpi.fict.ApplicationRunner$2.run(ApplicationRunner.java:89)
02-06-2020 13:58:17 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
02-06-2020 13:58:18 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
02-06-2020 13:58:19 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
02-06-2020 13:58:20 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
02-06-2020 13:58:21 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
02-06-2020 13:58:21 ERROR : Deadlock detected!
02-06-2020 13:58:21 ERROR : "Thread-0" prio=5 Id=16 BLOCKED on java.lang.String@290c0190 owned by "Thread-1" Id=17
02-06-2020 13:58:21 ERROR :     app//ua.kpi.fict.ApplicationRunner$1.run(ApplicationRunner.java:72)
02-06-2020 13:58:21 ERROR : "Thread-1" prio=5 Id=17 BLOCKED on java.lang.String@59fd348e owned by "Thread-0" Id=16
02-06-2020 13:58:21 ERROR :     app//ua.kpi.fict.ApplicationRunner$2.run(ApplicationRunner.java:89)
02-06-2020 13:58:22 INFO  : "Thread deadlock detector" prio=5 Id=14 TIMED_WAITING on java.util.concurrent.locks.Abst
02-06-2020 13:58:22 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
02-06-2020 13:58:23 INFO  : "Thread deadlock detector" prio=5 Id=14 TIMED_WAITING on java.util.concurrent.locks.Abst
02-06-2020 13:58:23 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
02-06-2020 13:58:24 INFO  : "Thread deadlock detector" prio=5 Id=14 TIMED_WAITING on java.util.concurrent.locks.Abst
02-06-2020 13:58:24 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
```

Рисунок 5.7 – Інформація у лог файлі

5.2 Випробування програмного продукту

Випробування програмного продукту:

- перевірити працездатність розкладів. Тобто перевірити, що сервіс на пошук негативних наслідків взаємодії паралельних процесів запускається кожні 5 секунд та сервіс пошуку інформації про потоки кожну секунду;
- перевірити працездатність сервісу пошуку негативних наслідків взаємодії паралельних процесів. Тобто перевірити, що сервіс знаходить негативні наслідки та дістає необхідну інформацію про ці наслідки (стан, код який потребує редагування, який об’єкт викликав негативний наслідок);

- перевірити працездатність сервісу пошуку інформації про потоки. Тобто перевірити, що сервіс шукає інформацію про всі потоки, крім системних та потоків, які мають негативні наслідки(стан, якщо потік у стані wait то який ресурс він чекає);
- перевірити працездатність виводу в лог файл. Тобто перевірити, що інформація виводиться у лог файл;
- перевірити коректність виводу інформації про негативні наслідки взаємодії паралельних процесів;
- перевірити коректність виводу інформації про всі потоки;
- перевірити ефективність роботи програмного комплексу;

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності функцій програмного комплексу технології автоматизованого виявлення негативних наслідків взаємодії паралельних процесів вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

- ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
- ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

1. Перевірка працездатності розкладів (рисунок 5.8).

```
02-06-2020 13:58:41 ERROR :
02-06-2020 13:58:41 ERROR :
02-06-2020 13:58:41 ERROR :
02-06-2020 13:58:41 ERROR :
02-06-2020 13:58:41 ERROR :
02-06-2020 13:58:41 INFO  :
02-06-2020 13:58:41 INFO  :
02-06-2020 13:58:42 INFO  :
02-06-2020 13:58:42 INFO  :
02-06-2020 13:58:43 INFO  :
02-06-2020 13:58:43 INFO  :
02-06-2020 13:58:44 INFO  :
02-06-2020 13:58:44 INFO  :
02-06-2020 13:58:45 INFO  :
02-06-2020 13:58:45 INFO  :
02-06-2020 13:58:46 ERROR :
02-06-2020 13:58:46 ERROR :
02-06-2020 13:58:46 ERROR :
02-06-2020 13:58:46 ERROR :
02-06-2020 13:58:46 ERROR :
02-06-2020 13:58:46 INFO  :
02-06-2020 13:58:46 INFO  :
02-06-2020 13:58:47 INFO  :
02-06-2020 13:58:47 INFO  :
02-06-2020 13:58:48 INFO  :
02-06-2020 13:58:48 INFO  :
02-06-2020 13:58:49 INFO  :
02-06-2020 13:58:49 INFO  :
```

Рисунок 5.8 - Перевірка працездатності розкладів

Отже, як бачимо з лог файлу, пошук негативних наслідків взаємодії паралельних процесів з поміткою «ERROR» логується кожні 5 секунд та пошук інформації про потоки з поміткою «INFO» логується кожну секунду.

Робимо висновок, що розклади працюють коректно.

2. Перевірка працездатності сервісу пошуку негативних наслідків взаємодії паралельних процесів (рисунок 5.9).

```
"Thread-0" prio=5 Id=16 BLOCKED on java.lang.String@290c0190 owned by "Thread-1" Id=17
    app//ua.kpi.fict.ApplicationRunner$1.run(ApplicationRunner.java:72)
"Thread-1" prio=5 Id=17 BLOCKED on java.lang.String@59fd348e owned by "Thread-0" Id=16
    app//ua.kpi.fict.ApplicationRunner$2.run(ApplicationRunner.java:89)
```

Рисунок 5.9 - Перевірка працездатності сервісу пошуку негативних наслідків взаємодії паралельних процесів

Отже, як бачимо сервіс знаходить блоковані потоки, показує їх стан, на якому об’єкті вони були заблоковані та строчку коду, де це сталося.

З цього робимо висновок, що сервіс пошуку негативних наслідків працює коретно.

3. Перевірка працездатності сервісу пошуку інформації про потоки (рисунок 5.10).

```
"Thread deadlock detector" prio=5 Id=14 TIMED_WAITING on java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@7cdeaf9b
"Thread state detector" prio=5 Id=15 RUNNABLE
"Thread deadlock detector" prio=5 Id=14 TIMED_WAITING on java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@7cdeaf9b
"Thread state detector" prio=5 Id=15 RUNNABLE
"Thread deadlock detector" prio=5 Id=14 TIMED_WAITING on java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@7cdeaf9b
"Thread state detector" prio=5 Id=15 RUNNABLE
"Thread deadlock detector" prio=5 Id=14 TIMED_WAITING on java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@7cdeaf9b
"Thread state detector" prio=5 Id=15 RUNNABLE
"Thread deadlock detector" prio=5 Id=14 TIMED_WAITING on java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@7cdeaf9b
"Thread state detector" prio=5 Id=15 RUNNABLE
```

Рисунок 5.10 - Перевірка працездатності сервісу пошуку інформації про потоки

Отже, як бачимо сервіс знаходить інформацію про потоки, їх стан, якщо потік у стані wait чому він чекає.

З цього робимо висновок, що сервіс пошуку інформації про потоки працює коректно.

4. Перевірка працездатності виводу в лог файл.

Як бачимо з попередніх рисунків (рисунок 5.8, рисунок 5.9, рисунок 5.10), що всі результати виводяться у лог файл.

Отже вивід у лог файл працює коректно.

5. Перевірка коректності виводу інформації про негативні наслідки взаємодії паралельних процесів.

Як бачимо з рисунку 5.9 інформація про негативні наслідки взаємодії паралельних процесів виводиться коректно.

6. Перевірка коректності виводу інформації про всі потоки.

Як бачимо з рисунку 5.10 інформація про всі потоки, крім системних та потоки, які мають негативні наслідки, виводиться коректно.

7. Перевірка ефективності роботи програмного комплексу (рисунок 5.11).

Кількість потоків	0	500	1000	1500	2000
Пошук інформації	0	0.022	0.034	0.045	0.074
Вивід інформації	0	2.281	15.156	49.628	116.742

Рисунок 5.11 - Перевірка ефективності роботи програмного комплексу

Як бачимо, що зі збільшенням кількості потоків, час на пошук інформації ледве росте, а на вивід інформації значно збільшується. Це тому, що вивід інформації процес громіздкий та займає багато часу.

Отже, програмний комплекс ефективний.

Висновок до розділу

В даному розділі були описанні основні функції, які були реалізовані та підтвердженні зображеннями екранних форм та наведений опис тестів і порядок їх виконання. Також було сформульована мета випробувань та наведенні результати випробувань. З результатів випробувань бачимо, що програмне забезпечення працює коректно і ефективно.

ЗАГАЛЬНІ ВИСНОВКИ

В ході виконання дипломного проєктування було визначено предметне середовище, були визначені призначення, цілі та задачі розробки.

В роботі був виконаний аналіз вхідних, вихідних даних, було проаналізоване математичне забезпечення. Також були створені вимоги до технічного забезпечення та було створене описання архітектури програмного забезпечення.

В ході розробки був використаний багатий інструментарій програмної мови Java, а саме клас ThreadMXBean, який має всю інформацію про потоки. Також для виведення інформації у лог файл було використано допоміжну бібліотеку log4j, яка дала можливість структурного запису у лог файл.

За результатами випробування програмного забезпечення зроблено висновок про коректну та достатньо ефективну роботу програмного комплексу.

ПЕРЕЛІК ПОСИЛАНЬ

1. YouKit Java Profiler [Електронний ресурс] – Режим доступу до ресурсу: https://www.yourkit.com/features/.
2. jProfiler [Електронний ресурс] – Режим доступу до ресурсу: https://www.ej-technologies.com/resources/jprofiler/help/doc/main/introduction.html.
3. VisuaVM [Електронний ресурс] – Режим доступу до ресурсу: https://visualvm.github.io/.
4. Алгебри процесів [Електронний ресурс] – Режим доступу до ресурсу:http://dspace.nbuv.gov.ua/bitstream/handle/123456789/50907/04-Nesterenko.pdf?sequence=1.
5. ThreadMXBean [Електронний ресурс] – Режим доступу до ресурсу: http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/management/ThreadMXBean.html
6. Executor Service [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/116363/.
7. Log4j [Електронний ресурс] – Режим доступу до ресурсу: https://coderlessons.com/tutorials/java-tekhnologii/uchitsia-log4j/log4j-kratkoe-rukovodstvo.
8. Процеси, потоки [Електронний ресурс] – Режим доступу до ресурсу: https://beasthackerz.ru/uk/skype/sozdanie-potokov-java-potoki-java-sozdanie-i-zavershenie.html.

Додаток А

Тексти програмного коду

Автоматизована технологія виявлення негативних наслідків паралельних процесів

(Найменування програми (документа))

DVD-R

(Вид носія даних)

50 арк, 1344 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації і управління

УЗГОДЖЕНО
Керівник проєкту
_____ *Інна СТЕЦЕНКО*
(підпис) (вл. ім’я, прізвище)
“13” квітня 2020 р.

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл. ім’я, прізвище)
“14” квітня 2020 р.

Програмний комплекс технології автоматизованого виявлення негативних наслідків взаємодії паралельних процесів

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 6120.01.000 ТЗ*

на 7 сторінках

Київ – 2020 року

Зміст

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування програмного комплексу

Програмний комплекс технології автоматизованого виявлення негативних наслідків взаємодії паралельних процесів.

1.2 Найменування організації-замовника та організацій-учасників робіт

Замовник програмного комплексу: ТОВ «Software Solutions».

Розробник програмного комплексу: студент 4 курсу ФІОТ, кафедри АСОІУ Охрименко Дмитро Юрійович.

1.3 Перелік документів на підставі яких створюється програмний комплекс

Підставою для розробки програмного комплексу є завдання на переддипломну практику.

1.4 Планові терміни початку і закінчення роботи зі створення програмного комплексу

Плановий термін початку робіт над програмним комплексом – 18.05.2020.

Плановий термін завершення робіт над програмним комплексом – 01.06.2020.

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Призначення програмного комплексу

Призначенням розробки даного програмного комплексу є зменшення проблем з коректним виконанням та завершення програм через недостатній аналіз й відслідковування негативних наслідків в програмах паралельного обчислення.

2.2 Цілі створення програмного комплексу

Ціль розробки: підвищити ефективність тестування паралельних програм за рахунок автоматизації виявлення негативних наслідків взаємодії паралельних процесів.

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Об'єктом автоматизації є процес виявлення негативних наслідків взаємодії паралельних процесів. Результат роботи досягається за декілька етапів:

- отримання детальної інформації про паралельні процеси;
- пошук взаємних блокувань взаємодіючих процесів;
- виведення інформації у лог файл;

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмний комплекс повинен виконувати наступні функції:

- виявлення негативних наслідків взаємодії паралельних процесів;
- виведення інформації про не цілком правильній код програми паралельних обчислень;
- виведення інформації про потоки(стан, тощо);
- виведення інформації про негативні наслідки, які виявлені;

4.2 Вимоги до надійності

Програмний комплекс не зачіпляє конфіденційної інформації програми паралельних обчислень, отже виток інформації неможливий

4.3 Вимоги до складу і параметрів технічних засобів

Мінімальні вимоги до характеристик комп’ютера:

- оперативна пам’ять – 8 Гб;
- процесор – Intel Core i5 2,9 МГц;
- графічна карта – Nvidia GeForce GTX 660;

Програмні засоби, що використовуються для проведення тестування:

- операційна система Windows 10, IDE Idea, вихідний код програми паралельних обчислень;

Технічні засоби, що використовуються для проведення тестування:

- оперативна пам’ять – 16 Гб;
- жорсткий диск – 500 Гб;
- процесор – Intel Core i5 8^{th} Gen;
- графічна карта – Nvidia GeForce GTX 780;

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Таблиця 5.1 – Етапи розробки

№	Назва	Результат виконання
1.	Постановка задачі	Технічне завдання
2.	Розробка математичної моделі	Розроблена математична модель
3.	Генерація тестових даних	Розроблено декілька програм паралельного обчислення з різною кількістю процессів та різною взаємодією процесів
4.	Розробка програмного комплексу	Розроблений програмний комплекс
5.	Тестування програмного комплексу	Протестований програмний комплекс
6.	Оформлення ПЗ	Готова пояснювальна записка

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Перелік документів, що повинні бути представлені на контроль:

- технічне завдання;
- пояснювальна записка опису програмного комплексу технології автоматизованого виявлення негативних наслідків взаємодії паралельних процесів;
- програмний комплекс технології фвтоматизованого виявлення негативних наслідків взаємодії паралельних процесів;
- тестові сценарії;
- керівництво користувача.

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

Дата перевірки:
12.06.2020 12:39:26 EEST

Дата звіту:
12.06.2020 12:50:48 EEST

ID перевірки:
1003989870

Тип перевірки:
Doc vs Internet + Library

ID користувача:
77149

Назва документу: Ohrimenko_is61_2

ID файлу: 1004003622 **Кількість сторінок:** 38 **Кількість слів:** 5880 **Кількість символів:** 43036 **Розмір файлу:** 102.23 KB

12.4% Схожість

Найбільша схожість: 4.85% з джерело бібліотеки. **ID файлу:** 1000017966

7.69% Схожість з Інтернет джерелами	26	Page 40
9.91% Текстові збіги по Бібліотеці акаунту	172	Page 40

0% Цитат

Не знайдено жодних цитат

0% Вилучень

Вилучений текст відсутній

Підміна символів

Заміна символів	1

Графічний матеріал
до дипломного проєкту

на тему: Програмний комплекс технології автоматизованого виявлення негативних наслідків взаємодії паралельних процесів

Київ – 2020 року

					ДП 6120.02.000 ССВ		
Зм.	Арк.	ПІБ	Підп.	Дата			
Розробн.		Охрименко Д.Ю.			Схема структурна варіантів використань	Літ.	Аркуш 1 / Аркушів 1
Керівн.		Стеценко І.В.					КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-61
Консульт.							
Н/контр.		Тєлишева Т. О.					
Затв.		Стеценко І.В.					

					ДП 6120.02.000 ССД	
Зм.	Арк.	ПІБ	Підп.	Дата		
Розробн.		Охрименко Д.Ю.			Схема структурна діяльності	Літ. / Аркуш 1 / Аркушів 1
Керівн.		Стеценко І.В.				
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-61
Н/контр.		Тєлишева Т. О.				
Затв.		Стеценко І.В.				

Зм.	Арк.	ПІБ	Підп.	Дата	ДП 6120.05.000 ССК		
Розробн.		Охрименко Д.Ю.			Схема структурна класів програмного забезпечення	Літ.	Аркуш 1 / Аркушів 1
Керівн.		Стеценко І.В.					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-61	
Н/контр.		Тєлишева Т. О.					
Затв.		Стеценко І.В.					

ДП 6120.05.000 ССК

					ДП 6120.05.000 ССК		
Зм.	Арк.	ПІБ	Підп.	Дата			
Розробн.		Охрименко Д.Ю.			Схема структурна компонентів програмного забезпечення	Літ.	Аркуш 1 / Аркушів 1
Керівн.		Стеценко І.В.					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-61	
Н/контр.		Тєлишева Т. О.					
Затв.		Стеценко І.В.					

```
21-05-2020 17:58:28 ERROR : Deadlock detected!
21-05-2020 17:58:28 ERROR : "Thread-0" prio=5 Id=16 BLOCKED on java.lang.String@7c1452d2 owned by "Thread-1" Id=17
21-05-2020 17:58:28 ERROR :     app//ua.kpi.fict.ApplicationRunner$1.run(ApplicationRunner.java:72)
21-05-2020 17:58:28 ERROR : "Thread-1" prio=5 Id=17 BLOCKED on java.lang.String@1a3276f1 owned by "Thread-0" Id=16
21-05-2020 17:58:28 ERROR :     app//ua.kpi.fict.ApplicationRunner$2.run(ApplicationRunner.java:89)
21-05-2020 17:58:34 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:58:35 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:58:36 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:58:37 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:58:38 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:58:38 ERROR : Deadlock detected!
21-05-2020 17:58:38 ERROR : "Thread-0" prio=5 Id=16 BLOCKED on java.lang.String@36f28163 owned by "Thread-1" Id=17
21-05-2020 17:58:38 ERROR :     app//ua.kpi.fict.ApplicationRunner$1.run(ApplicationRunner.java:72)
21-05-2020 17:58:38 ERROR : "Thread-1" prio=5 Id=17 BLOCKED on java.lang.String@65068b5c owned by "Thread-0" Id=16
21-05-2020 17:58:38 ERROR :     app//ua.kpi.fict.ApplicationRunner$2.run(ApplicationRunner.java:89)
21-05-2020 17:59:44 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:59:45 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:59:46 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:59:47 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:59:48 INFO  : "Thread deadlock detector" prio=5 Id=14 RUNNABLE
21-05-2020 17:59:48 ERROR : Deadlock detected!
21-05-2020 17:59:48 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
21-05-2020 17:59:48 ERROR : "Thread-0" prio=5 Id=16 BLOCKED on java.lang.String@6116879a owned by "Thread-1" Id=17
21-05-2020 17:59:48 ERROR :     app//ua.kpi.fict.ApplicationRunner$1.run(ApplicationRunner.java:72)
21-05-2020 17:59:48 ERROR : "Thread-1" prio=5 Id=17 BLOCKED on java.lang.String@79f33297 owned by "Thread-0" Id=16
21-05-2020 17:59:48 ERROR :     app//ua.kpi.fict.ApplicationRunner$2.run(ApplicationRunner.java:89)
02-06-2020 13:58:17 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
02-06-2020 13:58:18 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
02-06-2020 13:58:19 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
02-06-2020 13:58:20 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
02-06-2020 13:58:21 INFO  : "Thread state detector" prio=5 Id=15 RUNNABLE
02-06-2020 13:58:21 ERROR : Deadlock detected!
02-06-2020 13:58:21 ERROR : "Thread-0" prio=5 Id=16 BLOCKED on java.lang.String@290c0190 owned by "Thread-1" Id=17
02-06-2020 13:58:21 ERROR :     app//ua.kpi.fict.ApplicationRunner$1.run(ApplicationRunner.java:72)
02-06-2020 13:58:21 ERROR : "Thread-1" prio=5 Id=17 BLOCKED on java.lang.String@59fd348e owned by "Thread-0" Id=16
02-06-2020 13:58:21 ERROR :     app//ua.kpi.fict.ApplicationRunner$2.run(ApplicationRunner.java:89)
```